Ground Tools for Autonomy in the 21st Century

Kanna Rajan[1], Mark Shirley[2], William Taylor[3], Bob Kanefsky[1]

NASA Ames Research Center
MS 269-2
Moffett Field, CA 94035-1000
650-604-0573
{*kanna,shirley,taylor,kanef*}*@ptolemy.arc.nasa.gov*

Abstract—Ground tools for unmanned spacecraft are changing rapidly driven by twin innovations: advanced autonomy and ubiquitous networking. Critical issues are the delegation of low-level decision-making to software, the transparency and accountability of that software, mixed-initiative control, i.e., the ability of controllers to adjust portions of the software's activity without disturbing other portions, and the makeup and geographic distribution of the flight control team. These innovations will enable ground controllers to manage space-based resources much more efficiently and, in the case of science missions, give principal investigators an unprecedented level of direct control.

This paper explores these ideas by describing the ground tools for the Remote Agent experiment aboard the Deep Space 1 spacecraft in May of 1999. The experiment demonstrated autonomous control capabilities including goal-oriented commanding, on-board planning, robust plan execution and model-based fault protection. We then speculate on the effect of these technologies on the future of spacecraft ground control.

1. INTRODUCTION

Current methods of controlling robotic spacecraft rely on a relatively large and highly skilled mission operations team that generates detailed time-ordered sequences of commands to step the spacecraft through each desired activity. Each sequence is carefully constructed on the ground to ensure that all known operational constraints are satisfied. The autonomy of the spacecraft is limited.

An alternative approach to spacecraft operations was demonstrated in May 1999 by the Remote Agent Experiment (RAX) on the Deep Space One (DS1) spacecraft. In this approach, Artificial Intelligence technology is used to encode the operational rules and constraints in the flight software. The software may be considered to be an autonomous "remote agent" of the spacecraft operators in the sense that the operators rely on the agent to achieve particular goals.

For instance, operators often cannot know the exact conditions on the spacecraft so, under this new approach, they do not tell the agent exactly what to do at each instant of time. Instead, they tell the agent what high-level goals to achieve and what deadlines or other timing constraints there are on their achievement. The flight software then constructs and executes a detailed command sequence that reflects conditions onboard the spacecraft and obeys all constraints and flight rules. This is called "goal-oriented commanding."

The DS1 Remote Agent Experiment achieved multiple technology objectives. In addition to goal-oriented commanding, RAX demonstrated robust closed-loop plan execution. This enabled the remote agent to handle minor variations in the timing of spacecraft operations plus a broad class of minor problems without causing the planned command sequence to fail, e.g., if a switch did not turn on a device, try it a second time. In conventional commanding modes, had the device not turned on, the entire sequence would have failed, potentially safing the spacecraft. The experiment also demonstrated extensive fault protection capabilities, including failure diagnosis, failure recovery using both repair and reconfiguration, on-board replanning following otherwise unrecoverable failures, and system-level fault protection.

2. BACKGROUND

This section briefly describes the Remote Agent software architecture and the flight experiment. More detail can be found in [3]. Section 3 describes the ground tools created for RAX. Section 4 contains our analysis and predictions about the future of ground control.

[1] Caelum Research Corp., NASA Ames Research Center
[2] NASA Ames, Code IC
[3] Recom Technologies, NASA Ames Research Center

Software Architecture

The Remote Agent is formed by the integration of three separate Artificial Intelligence technologies: an on-board planner-scheduler (PS), a robust multi-threaded executive(EXEC), and a model-based mode identification and reconfiguration module (MIR).

The RA architecture and its relation to flight software are shown in Figure 1. Viewed as a black-box, RA issues commands to real-time execution flight software (FSW) to modify spacecraft state, and receives state information through a set of monitors that filter data streams into a set of abstract properties. The RAX manager mediated all communication between the Remote Agent and the DS1 flight sofware.

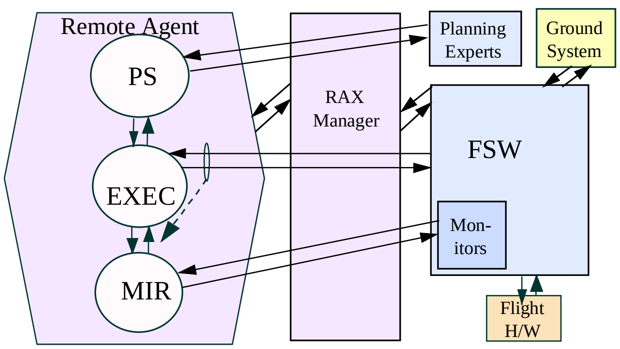

Figure 1: RAX Architecture

PS formulates near-term planning problems based on a long-range mission profile representing the goals of the mission, extracts goals for the next scheduling horizon, combines them with a projected spacecraft state provided by EXEC, and formulates a planning problem. This decomposition into long-range mission planning and shorter-term detailed planning enables RA to undertake an extended mission with minimal human intervention.

PS then takes a near-term planning problem and produces a flexible, concurrent temporal plan for execution by EXEC. PS constructs plans using domain constraints and heuristics in its knowledge base. Third-party software modules are interfaced to provide efficient computation for specialized problems such as navigational or attitude control which are beyond the scope of planner domain models.

EXEC executes a plan by decomposing the high-level plan activities into primitives, sending out commands, and monitoring progress based on direct feedback from the command recipient or on inferences drawn by MIR. If some task cannot be achieved, EXEC may attempt an alternate method or may request a simple recovery plan from MIR. If the EXEC is unable to execute or repair the current plan, it cleanly aborts the plan and attempts to bring the spacecraft into a safe state while requesting a new plan from MM.

MIR is responsible for mode identification (MI) and mode reconfiguration (MR). The MI task involves observing the EXEC as it issues commands, receiving events from monitors, and using a structural and behavioral model of the spacecraft to combine the sensor data and infer what state each component is in. MIR returns this information to EXEC producing the illusion that EXEC can observe the status of each component directly. During the mode reconfiguration task, MIR serves as a recovery expert. Using the same structural and behavioral models, it takes a set of EXEC constraints to be established or maintained as input and recommends a single step recovery action to EXEC. (This can be done repeatedly to generate a limited class of longer recovery sequences. For details of the Remote Agent see [7,8,9,10].)

The Flight Experiment

The flight experiment was conducted between May 17th and 21st 1999. The Remote Agent achieved all its technological goals and was fully flight validated with some problems encountered along the way. This timeline summarizes what occurred:

Oct 24 1998	DS1 Launch
May 17 1999	
11:04pdt	RAX starts First plan is generated (but with a deviation from testing runs) RAX performs nominally through the night
May 18	
7:00pdt	RAX team realizes EXEC has ceased to command the spacecraft
17:00pdt	RAX is stopped with 70% of validation objectives achieved
May 19-20	Problem debugged and software patch developed Completely new experimental scenario developed and tested on flight analog hardware in 10 hours Project team accepts the new scenario, rejects the patch
May 21	
7:30pdt	The new scenario is activated During the experiment, supporting software fails to confirm an IPS state transition (out of scope for RAX). Problem proves benign. RAX completes scenario achieving 100% of validation goals

The most significant events were the early deviation from planner testing runs, the discovery of the software bug, and the creation and testing of a new scenario in 10 hours (possible only because the autonomy software did much of the work). These events are described below. For a detailed account of the experiment, see [3].

Current Flight Control Software

The Jet Propulsion Laboratory's (JPL) ground tools are representative of the current state of the art. The widely used Multimission Ground Data System (MGDS) receives and stores telemetry from the spacecraft. The Data Monitor and Display Subsystem (DMD) allows ground controllers to read, analyze, and display specific telemetry and related channel values in a variety of formats. MGDS provides facilities for converting units and generating derived channels (functions of one or more other channels). Key parameters are then compared against pre-set limits and alarms are generated if those limits are violated. The data are then available to a variety of output ports such as tables, plots (channel vs. channel or channel vs. time), and alarm pages. The resulting user interface consists of sets of pages densely filled with color-coded numbers representing the state of the spacecraft.

The Remote Agent is fundamentally different from previous flight control software in the degree to which it (a) maintains an internal representation of what it is doing and why, and (b) infers the status of spacecraft components not observable directly by internal sensors. These differences are reflected in the ground tools.

3. THE RA GROUND TOOLS

The two major goals of the RA ground tools are:

1. Present a summary of the spacecraft status understood easily by the mission operations team.
2. Present enough information about the inner workings of the RA software for the experiment team to quickly recognize and debug problems.

To support these goals, telemetry specific to Remote Agent was downlinked during the test. Given the abstract nature of the RA decision-making, conventional telemetry describing the spacecraft state would have given only the coarsest view of the Remote Agent's performance. RA could potentially generate correct commands (visible via conventional telemetry) for the wrong reasons (visible only with RA-specific telemetry), which would not bode well for the remainder of the test.

The RA-specific telemetry included contained:

1. Planning events (e.g., planning started, finished, and progress messages)
2. Sequence execution events (e.g., plan `p` is starting execution, or plan step `x` started executing at time `t1`)
3. Mode interpretation events (e.g., valve `v` changed state from `nominal` to `possibly-stuck-closed`)
4. Messages between RA components and between RA and other DS1 flight software.

This telemetry was event oriented and largely incremental, i.e., interpretation of one message depended upon receipt of earlier messages. This design created some problems we will detail later. In addition, there were heartbeat ("I'm still alive") messages from RA components visible in the normal DS1 telemetry. The next sections describe specific ground tools constructed to examine this telemetry.

Telemetry Data Flow

Figure 2 shows how this telemetry flowed from the spacecraft to the RA ground tools. First, downlinked telemetry and files were received by the Deep Space Network (DSN) and sent to MGDS. Remote Agent ground software (mgds2rax and the Decoder) then retrieved RA-specific telemetry from the MGDS database and published packets on a software message bus (TCA-IPC) used by all RAX ground tools. The flexibility of this message bus enabled us to run duplicate collections of ground tools at JPL and NASA Ames and to safely add modified versions of our ground tools at the last minute to support the RAX web site, further illustrating the plug-and-play paradigm.

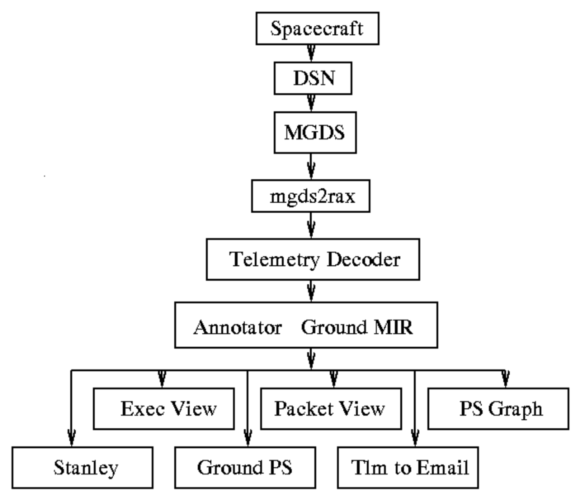

Figure 2: RA Tool Data Flow

Packet View

Remote Agent telemetry is event-based.[4] PacketView displays these events in a simple one line per message format that is easily understandable to all members of the team. As such, PacketView was the foundation of the RA ground tools (see Figure 3).

---

[4] This caused problems with dropped and delayed packets as we describe later.

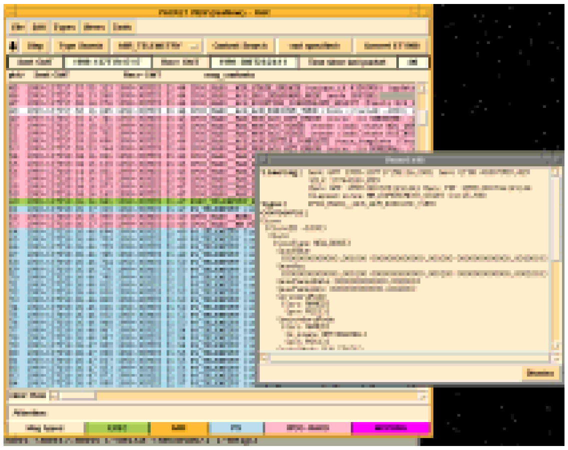

Figure 3: PacketView (telemetry packet) display

The PacketView interface displayed four types of telemetry packet: EXEC - executive telemetry, PS - planner/scheduler telemetry, MIR - mode identification/ recovery telemetry, and IPCO_RAXO - on-board messages sent between the three RA components. These were displayed as color-coded scrolling lines of text. Missing packets, detected by gaps in the sequential packet numbering, were displayed as a single line in a unique color. Search capabilities were available based on message type and content. User selectable dialogs presented "pretty-printed" versions of the single-line packet entries. The "time bar" displayed the most recent "spacecraft sent" Greenwich Mean Time (GMT), the most recent "ground received" GMT time, and a running time since last packet was received.

ExecView

ExecView visualized the execution status of plans onboard the spacecraft (see Figure 4). Different kinds of activity associated with different parts of the spacecraft appeared on separate timelines. For each kind of activity, specific planned events were represented as "tokens" appearing along the timeline. Tokens were color-coded to represent their execution status, i.e., in-the-future, currently-executing, completed and completion-overdue.

The Remote Agent used a constraint posting planner that models the activity to be scheduled as a set of tasks with temporal constraints among them (see [18]). For instance, communication with the ground couldn't occur during an interval in which the spacecraft was turning to point its camera at a target. It was the job of the planner to find a schedule that satisfied the many constraints between activities. A significant part of understanding the performance of the planner involved understanding its handling of these constraints. ExecView provided ways to visualize these temporal constraints, and the user could see how they evolved as the plan execution proceeded.

As the plan was being executed by Exec onboard the spacecraft, the start and finish times of the activities would be expected to change. Through the constraints, these changes would impact later activities. ExecView would propagate these changes downstream in the schedule, using the same propagation techniques used by the Planner.

ExecView was designed initially as a debugging tool for validating Exec development. As a result it did not have support for handling missing telemetry packets during flight. As a result, it produced some erroneous conclusions during RAX concerning the state of plan execution. To make ExecView more useful, it will have to handle such missing data. One approach would be to view the problem ExecView is solving as a dynamic constraint satisfaction problem (see [19]) where some terms are unknown during testing and other terms are unknown during flight. In testing, the Exec's actions and the current clock time are known completely, and the unknown is whether the EXEC is operating correctly (executing tokens at the correct times). In flight, the reports of actions taken and the knowledge of the current clock time may be incomplete, but it can assume that the EXEC is doing its job. If the EXEC reports Task B5 is finished after a few missing packets, ExecView could safely assume that Task B5 was previously started, and even that Task B4 must have finished.

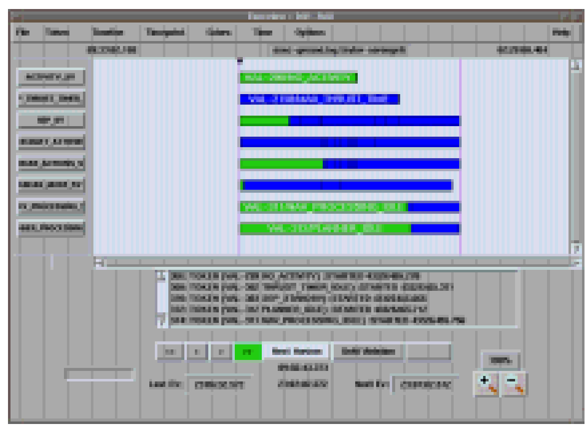

Figure 4: ExecView (plan execution status)

The Ground Planner

Of the three technology modules flown as part of the RA, the spacecraft team was least comfortable with PS. To allow the DS1 team to gain confidence in the planner, the RAX team used a ground twin. The ground planner was identical to the one onboard and was able to duplicate the onboard twin by tapping into the real time telemetry available on the ground. It had access to other flight software resources via connection to a high-fidelity testbed. This testbed accurately replicated the software onboard DS1 although it did not replicate the hardware. Of particular importance to the planner were navigation module and beacon asteroid file describing targets for optical navigation and the portion of the attitude control software that predicted the time required to change spacecraft orientation.

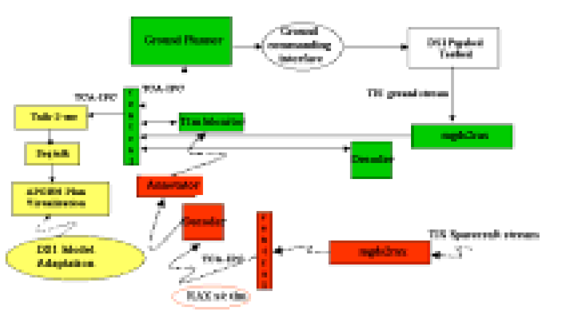

Figure 5: Ground Planner Twin system architecture

With one exception, described below, the ground planner was a useful tool in predicting the performance of the planner onboard and was especially useful as a confidence builder for mission staff unfamiliar with the working of an autonomous planning agent. Figure 5 gives an overview of the system architecture of the ground twin.

PS Graph

PS Graph (see **Figure 6**) displayed search history for each plan generated by the on-board Planner as a x-y graph representing the search depth vs. number of search nodes visited. The purpose of these plots was to provide a gross metric indicating whether the planner was proceeding as it had during testing and to check agreement with the ground twin.

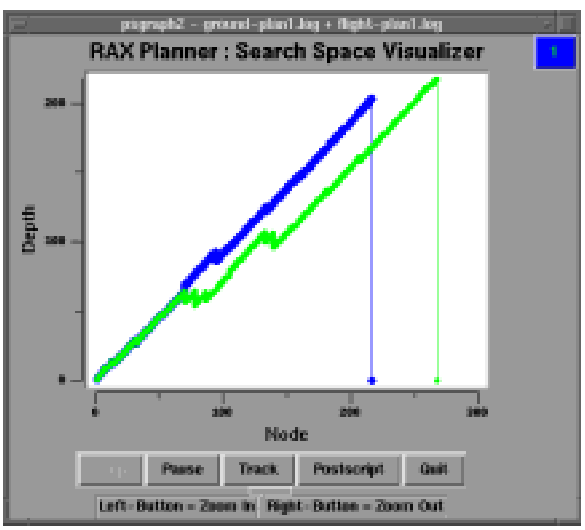

Figure 6: PS Graph (planner progress) display

During flight, we noticed that the onboard planner's search history deviated from both the ground twin and previous testing. Since the software and file configurations were expected to be identical between the ground and the spacecraft, this discrepancy was unexpected. We later discovered the discrepancy to be caused by a difference in the content of the file containing the onboard navigator's beacon asteroids. This file version difference resulted in a navigation plan involving different asteroids, attitudes and turn times. Though designed for just this sort of variation, the planner onboard was, in fact, asked to solve a problem it had never been tested on! And it did so successfully. This unintentional demonstration gave the flight team greater confidence in the new planning technology.

Stanley and MIR

A version of MIR was also run on the ground. The purpose of this was to infer MIR's full internal representation of the spacecraft state from the telemetry which contained a much smaller subset. Specifically, it contained the set of independent variables in MIR's spacecraft model. The Stanley[5] ground tool displayed a hierarchical schematic of the spacecraft's on-board components whose status was driven by the ground MIR (see **Figure 7**).

Components could be opened to show more detail or closed. The states displayed were blue (ok - powered off), green (ok - powered on), yellow (recoverable failure), purple (degraded failure), and red (permanent failure). Since Stanley assigned colors to all states, nominal as well as off nominal, it allowed the user to tell at a glance the conditions of the devices. Stanley did not address the issue of displaying continuous values, such as a battery state of change. DMD displayed such values textually, with units.

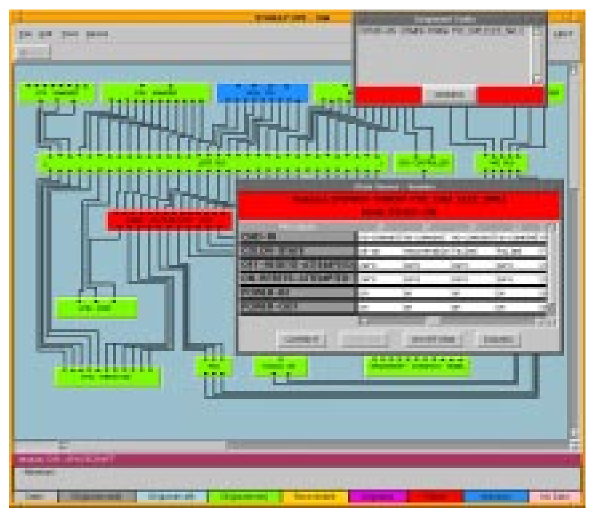

Figure 7: Stanley (hardware status) display

In addition to the color changes, detected component faults were reported by popping up an alert box. The alert box allowed the user to click on an entry, resulting in the sche-

[5] The engine underlying MIR is called Livingstone

matic being opened down to the appropriate hierarchical level to show the local context of the fault. Histories of all state changes, important or not, were available at any time by clicking on components.

Predicted Events

Although the Remote Agent is autonomous in principle, ground operators need to coordinate certain activities with it. For example, the Deep Space Network stations need to know when to expect signals from the spacecraft and when to expect its High Gain Antenna to be pointed away. Also since ground operators were watching the commands RA issued closely, it was important to have a prediction of when the RA planned to take various actions, so that the appropriate subsystem stations at the mission operations center could be staffed. Like the conventional MGDS system, the RA ground system produced a predicted events list of “activities scheduled for execution”. However, the activities for RA were high-level and structured, the scheduling was done onboard and was flexible, and the execution was more complex.

In conventional missions, commands are prepared on the ground, and time specifications provided by the operations and science teams (e.g. “start this command ten seconds after the previous command starts”) are reduced to absolute time values. A Predicted Events File (PEF) is a fully expanded version of the low-level command sequence, along with calls to on-board routines and the like. This is generated for use during the final command approval process.

In the case of the RA, a plan is generated on board from the goals outlined in the mission profile, and so the timing is not known until the plan is reported via telemetry. Even then, there is flexibility in the timing since only as an activity is being executed will specific times for its initiation and termination be chosen. Until execution, activities are associated with time windows. For instance, while a traditional sequence will allow a fixed amount of time for the Xenon tank to pressure before turning on thrust, the Remote Agent can give the next command any time within a flexible time window, as soon as it finds that all preconditions for thrusting are met. The predicted-events printouts used for RA therefore indicated the earliest and latest possible start times.

The printouts reported both the low-level commands RA would issue together with the high-level actions RA was asked to take. The high-level actions served as a summaries for and explanations of the low-level commands. RA was designed to decide on task decomposition in real time, not to report them in advance, so the breakdown and timing were exacted from logs of pre-flight tests. In this case, the low-level advance information was of interest only because of the experimental nature of the RA, but there may inherently be an issue to be kept in mind at design time: as in human planning, some details that can be decided on at the last moment may have an impact on a third party's own planning and so may need to be settled in advance.

```
07:07:58 Rotating the spacecraft, to point MICAS camera at Asteroid 39 Laetitia, and the solar panels
         at the sun.
07:21:16 Taking an image of Asteroid 39 Laetitia, as requested by Autonomous Navigation system.
07:44:52 Rotating the spacecraft, to point the ion engine thrust at ra 4.1666, dec -0.3550, x=-0.4867,
         y=-0.8014, z=-0.3476, and the solar panels at the sun.
09:00:56 Commanding the ion propulsion system to pressurize the xenon tank.
09:11:12 Commanding the ion drive to start thrusting.
09:11:15 Commanding Attitude Control Subsystem to use ion drive for steering.
12:54:10 Commanding Attitude Control Subsystem to use thrusters for steering.
12:54:13 Commanding the ion drive to stop thrusting.
16:09:11 Rotating the spacecraft, to point High Gain Antenna at Earth, and the solar panels at the sun.
16:11:51 Spontaneous spacecraft state changes just detected:
        - Steering with thrusters is degraded but usable.
        - Earth standby capability is degraded but usable.
        - Sun standby capability is degraded but usable.
        - Ability to use X-facing thrusters for Y-axis torque has failed and appears unrecoverable.
        - Ability to use Z-facing thrusters for Y-axis torque is degraded but usable.
16:11:51 RAX is looking for a way to recover. The recovery will have to achieve (or maintain) four im-
         portant states:
        - Make it so steering with thrusters is working okay.
        - Make it so ACS is steering with thrusters.
        - Make it so MICAS camera is powered on.
        - Make it so Low-Power Electronics subsystem is working okay.
16:11:52 In an attempt to recover, RAX has decided to command Attitude Control Subsystem to employ its
         ability to use Z-facing thrusters for Y-axis torque.
```

Figure 8: Sample of summarized telemetry

Public Outreach via the Web

The RAX team constructed two additional tools to present the remote agent's progress to the public. These tools are interesting because they required an even higher target for simplicity and understandability than did the flight controllers's tools. The tools were emailed summaries of events onboard presented in simple English and a Java applet timeline display patterned after `execview`

Several recent missions have used pagers and email to deliver notifications to the mission operations team. The DS1 ground system, for instance, alerted operators by pager when a given measurement strayed outside a preset range or when fault protection telemetry went into an unusual state. We took this a step further in RAX by producing descriptions of important events in common English. See figure 8 for an example. The summarized descriptions were automatically posted to our web site (rax.arc.nasa.gov) and emailed in batches to a public mailing list. 2000 subscribers received this email during RAX. Terse descriptions were also sent to team members' alphanumeric pagers via email.

This was done by applying a set of rewrite rules to the RAX telemetry. These rules filtered the event stream to identify important events. They then expanded the stylized event descriptions into English sentences using templates and translated event parameters using lookup tables. For example, a DS1 turn specification includes about a dozen parameters including an asteroid id number and codes for the antenna and camera bore sights. These numbers were looked up in tables to produce human-readable messages.

Critical to achieving event summaries that would convey what the Remote Agent was doing onboard, without obscuring detail, was Remote Agent's ability to represent its own goals. This allowed us some flexibility in tuning the event filtering to provide an appropriate level of detail. Even then, we didn't achieve English summaries as effective as we wanted because some information about why RA was doing certain actions was not downlinked during the test but stored onboard for later transmission. Were we to do this again, we would include more of this information in telemetry.

We also provided an alternative description of Remote Agent activity (Figure 9) using horizontal timelines patterned after ExecView. This was implemented as a java applet. The timelines in the top window represented major kinds of activity (e.g., attitude or camera-related activity). Along the timelines were boxes indicating particular activities (e.g., a turn) in effect reproducing the plans generated onboard on a user's web browser. At the bottom left were controls to step through the timelines. At the bottom right was an event-based summary similar to that provided in email. This box was slowly added to as new telemetry came down. The most interesting feature of this applet was its ability to show what RA planned to do at any point in the experiment by selecting the event that occurred at that time. This is interesting because the plan changed several times due to simulated faults. Thus it provided an historical overview of Remote Agent's re-planning activity and recreated for the general public conditions onboard the spacecraft.

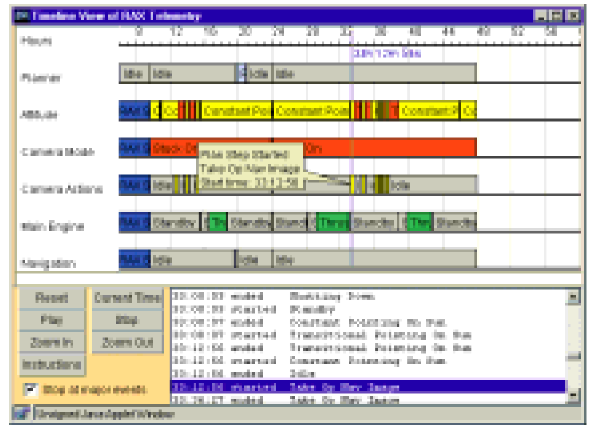

Figure 9: Timeline Applet

Due to time pressure, the outreach tools were designed to handle the nominal scenario only (including the simulated faults). They did not accurately reflect the RAX software problems that occurred. They did, however, summarize activity during the new scenario without modification.

These outreach tools caused some debate within the project team because of their potential to inform the public immediately of a problem onboard before it was well understood by the team. It was always assumed the public would get this information, but the possibility that the tools would magnify the public's awareness of small problems or problems that could easily be handled was a legitimate worry. Also, bugs in these explanation tools that gave a false impression of activity onboard couldn't be easily corrected if summaries were sent automatically. In the end, we believe the team allowed the tools because RAX was an experiment that did not risk the spacecraft. In the future, we will continue enhancing our tools for summarizing and presenting events onboard spacecraft to the operations team. How this information best gets to the public is still open.

4. DISCUSSION

High-level commanding & delegation

Central to Remote Agent is the idea of ground controllers delegating tasks to a software agent onboard. What distinguishes RA from sequencing software is the degree to which the tasks can be specified in terms of high-level goals, leaving to the agent the details of accomplishing the goals and managing their interactions.

Delegation to a software agent has several advantages. First, if reliable, delegation can reduce ground controller workload significantly. Second, the agent can handle contingencies and off-nominal events quickly, without the delays associated with safing the spacecraft and communicating with Earth and thereby avoiding regeneration, testing and validation of possibly complex sequences. In time-critical phases of a mission, for instance Cassini's insertion into Saturn orbit, handling contingencies quickly is not just essential but critical to the success of the mission.

A simple version of such a scenario, but one that occurs more often in practice, is handling minor deviations in plan execution such as steps finishing later than expected. This is possible when the agent "understands" the relationships between different steps in the plan. For instance, a delay is normally not a problem, except when it in turn delays a later step with absolute time constraints, say for instance, a communication window with the DSN. Deciding which situation the agent is in and compensating if necessary requires reasoning about these relationships. Note that timing variations can be helpful too, such as when an activity finishes earlier than planned for. This may leave extra time for other activities, such as science observations that may otherwise not be optimized.

High-level commanding has a profound impact on ground tools. Clearly they must be able to display goals, plans and plan execution status. And they must indicate deviations from the plan and any corrective actions the agent takes. High-level commanding also makes possible summaries like those generated by the telemetry-to-email and html tools. Central to doing this in a general way are the relationships between the high-level goals and low level commands used by the agent to achieve them. These relationships allow a ground tool to cluster the low level commands and summarize them as a single event. Of course, the details must be easily available. There is a fundamental tension between human operators, with their experience, intuition and sophisticated pattern recognition abilities, focussing on raw sensor data from a spacecraft and the associated high workload versus the pressure to handle more missions with fewer resources. We believe summarization of this sort is an effective way for ground operators to avoid information overload in the future.

Yet another aspect of such a paradigm has to deal with missions of extended duration. As humans have gone from exploring celestial bodies closer to earth, to those further away, the time it would take to get there is going to increase. At the same time, budget constraints have forced space agencies to achieve ambitious goals within smaller operations teams. One prominent example of such diverse goals in a mission is the COmet Nucleus TOUR (CONTOUR) mission to be run out of the Johns Hopkins Applied Physics Laboratory. The basic goal is to visit 3 comets in different stages of evolution over a 6 year mission (see Figure 10). With the exception of the time during encounters, CONTOUR will be in a spin stabilized hibernation mode. . During the non-encounter periods of the mission the spacecraft systems will be put into a hibernation mode and the operations staff will be very low or non-existent. At encounter – 60 days the operations staff will be reactivated, contact will be made with the spacecraft, and a checkout will begin. Because the mission will be intermittently staffed, it is highly likely that spacecraft knowledge by the operations staff would have degraded and have to be regained when the spacecraft awakes from its hibernation. In addition it is difficult to retain detailed knowledge of previous states and behavior of the spacecraft and its subsystems. Either the necessary personnel are no longer available or they have moved on to other projects and are no longer experts for the current mission.

Under such a situation, ground tools which can rapidly provide a quick overview of the spacecraft using notations based on high level commanding would be most valuable. Missions staff unfamiliar with the modalities of operating a different mission or craft can quickly come up to speed and achieve situational awareness in a matter of days and not months. They can be provided with high level commanding goals enabled on the spacecraft during its lifetime and understand the nature and cause of faults encountered onboard. Using conventional means, the user would be forced to delve into large datasets to fully understand the scope and history of the past performance of the spacecraft not to mention be unprepared to deal with any contingencies within the current timeframe.

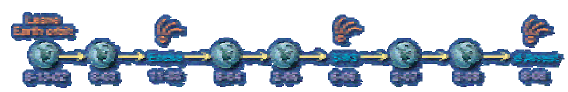

Figure 9: The CONTOUR Mission

Science Operations

High-level commanding has benefits outside of the operations community. Applied to a science mission, the logical extension of the idea is for the science team to interact with the onboard software agent directly through high-level goals and summarizations of the resulting activity (plus the resulting science data). The operations team would focus on interaction with shared resources like the DSN and on problems when they arose. We did not explore this with the remote agent experiment, which was primarily a technology demonstration. Our team did not interact directly with anyone who sought space science instrument data.

On Earth, however, our research group has done a series of autonomous rover field tests with the goal of making the rover increasingly an extension of the scientists' eyes and hands. These tests have used high-level commanding and delegation to a software agent as a way to enable the scientists to interact directly with the rover in terms they care about and avoiding technical details that can be abstracted and handled automatically. With the expected increase in the number of NASA and commercial missions, we expect an increased emphasis on Science Operations. Mission Operations, we believe, will become more opaque to the actual end users. Networking will also enable PI's to control missions from their home institutions in distributed geographic locations rather than gather for extended periods at a mission control center.

Mixed-Initiative Control

New generations of autonomy technology like Remote Agent will provide the ability to handle more complex missions more robustly. But will operations and science teams accept the reduced level of ground control inherent in higher levels of onboard autonomy? Are there risks to increased autonomy?

Yes there are: autonomy is a tradeoff. Where are the checks and balances for decisions made onboard? What about reduced situational awareness? In a time-critical situation, will ground operators have difficulty getting "up-to-speed" on the state of a spacecraft that has been monitored and adjusted by an onboard agent? During science missions, investigators are rightly concerned about any filtering or interpretation of instrument observations onboard or any automated focus-of-attention mechanisms. Note that all of these issues also arise in the relationship between astronauts and ground controllers.

Although RAX was intended as a demonstration of full autonomy, we recognize these legitimate concerns and are working to provide variable levels of autonomy. Remote Agent as it was tested could not do this. It was all-or-nothing: either it was responsible for managing tasks within its prescribed area of activity or not. Our research group at NASA has several projects underway to examine various ways to allocate responsibility. The key here is the notion of "mixed-initiative" control. This means the ability to seamlessly handle control actions initiated both on the ground and within the onboard agent. The key problems are making sure that activities commanded by the agent and the ground do not conflict with each other or incrementally replanning any agent-initiated activity that does conflict; designing user interfaces that convey to the ground operators the effects ground commanded actions will have on the currently planned onboard activities; and ground tools to assist operators in constructing their commands quickly and accurately.

While autonomy should enable more complex and robust missions, it will also reduce controller workload allowing controllers to handle more spacecraft. This will be critical to achieve a new kind of mission: the spacecraft constellation.

Constellations

Spacecraft constellations have recently become a hot topic driven by many factors including the distributed interferometry techniques planned for the terrestrial planet finder and NASA's cheaper, faster, better design philosophy. SensorWeb is one long-range concept proposed by the Goddard Space Flight Center in which hundreds of heterogeneous satellites cooperatively achieve our Earth observing goals. Redundancy and the ability to incrementally upgrade the constellation with new instruments without significantly changing the control problem are two advantages of this concept. Near-term efforts include The New Millennium ST3 with its plan to fly two similar spacecraft within a tight formation to image identical portions of the sky. JHU/APL is also expecting to fly the STEREO mission to study solar activity and provide data in real-time to understand global climactic changes on earth due to this phenomena.

Constellations are here to stay. One can expect to see constellations of cheap deep space probes being sent out to discover life in other solar systems. These could be homogenous or heterogeneous and controlled using a classical master/slave control law or in a more distributed manner. In any case, the complexity of performing mission operations increases dramatically with additional platforms in space however simple they might be. High level commanding and transparent control will not only be desired but be necessary for such mission design. In the event of a collective anomaly (massive solar flares for instance), operators will need to rapidly assess the state of the constellation fleet and access damage and react to save multiple spacecraft that might have reacted in different ways. Their tools will not only have to provide a succinct high-level view of the situation but provide details of individual spacecraft and suggest potential recovery actions. We believe technologies such as the Remote Agent, which works at higher abstraction levels, has an inherent advantage over the suite of tools currently available. Moreover, portions of these constellations may be deployed and owned by different organizations and countries. The desire to coordinate commanding across organizations will place an even higher premium on tools for maintaining situational awareness across the whole constellation.

The same can be said of doing science. With greater numbers and diversity of platforms a principal investigator would need to manipulate simultaneously for doing say interferometry, it is inconceivable that s/he would be in a position to command and control such multitudes of vehicles without either a vast array of operations staff or tools which will abstract telemetry data and provide a clear picture of the state of the system.

Verification and Validation

Section 2.2 of this paper gives a brief synopsis of the Remote Agent in flight and mentions the discovery of a bug leading to the stoppage of the experiment when 70% of the validation objectives had been accomplished.

It is instructive to look closely at the way the bug was diagnosed and conclusively discovered. A missing critical section in a part of the Executive's code was the reason one Exec thread was waiting for the other indeterminately. The bug itself was noticed after it was discovered that the RA had stopped executing, but that the spacecraft state itself was deemed healthy. In order to get conclusive evidence as to the nature of the bug, members of the RA team uploaded sequences that evaluated variables in the running environment of the Exec. The returned values indicated the state of each of the concurrent threads the Exec was supposed to be running which helped in rapid convergence to the portion of the Exec code where the missing critical section was found. A quick test with various timing parameters to simulate onboard conditions revealed that this was indeed the case. Simultaneously, while the debugging was being undertaken at JPL, members of the Automated Software Engineering group had been challenged to find the same bug in the RA software without access to any run time data. Indeed, Verification and Validation (V&V) techniques used by the group independently found the bug on a simplified model of the Executive's plan runner module which does thread management. It turns out a similar bug was found by them 2 years earlier (see [16]) in a different section of the code; the flight bug was a cut and paste error which was not corrected when the first error was found and fixed!

The events described above highlight the importance of V&V techniques in debugging complex software. While they are not a panacea in themselves due to their inability to completely explore all possible paths of flow of complex systems, techniques are available to model restricted subsets of the execution traces of the software itself. Such meta-models allow limited (albeit useful) evaluation such as the above, to take place. As a result of this exercise, therefore, more research is currently being undertaken at NASA Ames to fully understand the scope of domain models of spacecraft and to automatically model them within model-checking tools such as [17]. Increased emphasis on automated scenario based testing [8] is also critical to understand how software for complex autonomous systems is likely to behave in a situated environment and to provide some measure of guarantee and reliability in mission critical environments.

Spacecraft as Appliances

Increasingly in the Internet world, the concept of appliances are becoming a reality [4]. For instance internet accessible vending machines that check the status of their contents have been available for years [1]. Web based control interfaces to devices like robots and cameras have also become ubiquitous. Microsoft, the U.S. Geological Survey and others have come together to provide satellite imagery from the SPIN-2 spacecraft in 2-meter resolution via the web as part of the TerraServer project. Some of the applications touted from this venture are in discovering and fighting forest fires, assessing environmental damage, urban planning, planning for construction sites and just for "plain fun" [5].

Given this trend, it's very possible that people would like to have space imagery for prospective vacations, for navigation in unfamiliar surroundings and simply as a means to understand the topography of a particular area. We predict there will come a time when spacecraft themselves will also become appliances which will be accessed by ordinary citizens for purposes of science, entertainment or perhaps even as proxy for traveling out in interstellar space. The spacecraft in effect becomes a "server" for an individual's needs, whether it is for science or entertainment.

While simple interfaces will suffice for requesting and obtaining space based images like those by the TerraServer, for individuals to be able to remotely control and maintain spacecraft as entities in space will require more than the simple interfaces available. Light time communication delays will increase the importance of agents acting as proxies for groups of users on the ground, representing their interests and goals as conditions change in space. Symbolic representations of spacecraft data will be essential so that data is easily understood, digested and manipulated by lay individuals. Such an individual will require abstractions of subsystems now commanded to the minutest detail, like ACS, OpNAV and engines. They will need abstractions to manipulate the orientation of the spacecraft for optimal imaging patterns without bringing harm to the craft. Abstractions of telemetry data received by the individual will provide high-level information to the user at a glance, and will enable users to dig down into details when necessary.

5. CONCLUSION

Advanced autonomy is providing new opportunities for flight controllers and science teams to manage unmanned spacecraft more efficiently. Autonomy allows delegation of low-level decision-making to software, more rapid generation of verified command sequences, and is a critical enabler for certain robotic mission scenarios that cannot be controlled effectively from the ground. However, improving this category of software's transparency, accountability and capability for mixed-initiative control is critical to its acceptance by the flight control community. We have explored these issues via the ground tools for the Remote Agent experiment onboard the Deep Space 1 spacecraft in May of 1999.

ACKNOWLEDGEMENTS

This paper describes work performed at the NASA Ames Research Center and at Cal Tech's Jet Propulsion Laboratory under contract from NASA. The Remote Agent and its ground tools were the work of a large team of researchers and developers from both institutions.

REFERENCES

[1] "bsy's List of Internet Accessible Coke Machines", at http://www.cs.cmu.edu/afs/cs.cmu.edu/usr/bsy/www/coke.html.

[2] Bernard D., et al., "Design of the remote agent experiment for spacecraft autonomy." In Proceedings of the IEEE Aerospace Conference, 1998.

[3] Bernard, D. et al, Spacecraft Autonomy Flight Experience: The DS1 Remote Agent Experiment (AIAA?)

[4] Eustice, K., et.al, "A universal information appliance", IBM Systems Journal, Vol. 38, No. 4, 1999

[5] Microsoft TerraServer, http://www.terraserver.microsoft.com.

[6] Muscettola N., "HSTS: Integrating planning and scheduling." In Mark Fox and Monte Zweben, editors, Intelligent Scheduling. Morgan Kaufmann, 1994.

[7] Muscettola N., P. Nayak., B. Pell, and B. Williams, "Remote Agent: To Boldly Go Where No AI System Has Gone Before," Artificial Intelligence 103(1-2):5-48, August 1998

[8] Nayak P. et al, "Validating the DS1 Remote Agent Experiment," Proceedings of the 5th International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS-99)

[9] Nicola Muscettola, P. Pandurang Nayak, Brian C. Williams, and Barney Pell, "Remote Agent: To boldly go where no AI system has gone before." Artificial Intelligence, 103:5-47, 1998.

[11] Pell B, et al, "An Autonomous Spacecraft Agent Prototype," Autonomous Robotics, 5(1), March 1998.

[12] Pell B., et al, "Robust periodic planning and execution for autonomous spacecraft." In Proceedings of IJCAI-97, 1997.

[13] Remote Agent web page http:://rax.arc.nasa.gov

[14] Smith B., et al, "Validation and verification of the remote agent for spacecraft autonomy." In Proceedings of the 1999 IEEE Aerospace Conference, 1999.

[15] Williams, B. and P. Nayak, "A model-based approach to reactive self-configuring systems." In Proceedings of AAAI-96, pages 971-978, 1996.

[16] Havelund, K, M.Lowry and J. Penix "Analysis of a Spacecraft Controller using SPIN", In Proc. 4th International SPIN Workshop, Paris, France, Nov 1998.

[17] Bengtsson,J. et.al, "UPPAL - a Tool suite for Automatic Verification of Real Time Systems", In Proc. 4th DIMACS Workshop on Verification and Control of Hybrid Systems"", New Brunswick, New Jersey, Oct 1995.

[18] Jonsson.A, P.Morris, N.Muscettola, K.Rajan and B. Smith, "Planning in Interplanetary Space: Theory and Practice", submitted to the AI and Planning Systems conference, Brekenridge, CO April 2000.

[19] Dechter, R & A. Dechter, "Belief Maintenance in Dynamic Constraint Networks", in Proc. 7th National Conference on Artificial Intelligence, AAAI-88, Palo Alto, CA.